ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **15**, Number 2, 2012, 106–128

Dynamic Selection of Enumeration Strategies for Solving Constraint Satisfaction Problems

Broderick CRAWFORD[1], Carlos CASTRO[2], Eric MONFROY[3], Ricardo SOTO[1,4], Wenceslao PALMA[1], Fernando PAREDES[5]

[1] Pontificia Universidad Católica de Valparaíso, Chile
[2] Universidad Técnica Federico Santa María, Chile
[3] CNRS, LINA, Université de Nantes, France
[4] Universidad Autónoma de Chile, Chile
[5] Escuela de Ingeniería Industrial,
Universidad Diego Portales, Santiago, Chile

Abstract. The main goal concerning Constraint Satisfaction Problems is to determine a value assignment for variables satisfying a set of constraints, or otherwise, to conclude that such an assignment does not exist (the set of constraints is unsatisfiable). In the Constraint Programming resolution process, it is known that the order in which the variables are assigned have a significant impact in terms of computational cost. In this paper we propose a new framework for guiding the classical Constraint Programming resolution process. Such a framework is able to measure the resolution process, using some indicators, in order to perform on-the-fly replacements of variable/value ordering heuristics exhibiting poor performances. The replacement is performed depending on a quality rank, which is computed by means of a choice function-based Hyper-heuristic, where its parameters are fine-tuned by a Genetic Algorithm which trains the choice function carrying out a sampling phase. The experimental results show the effectiveness of our approach where our combination of strategies outperforms the use of individual strategies.

1. Introduction

Constraint Programming (CP) is a powerful software technology devoted to the efficient resolution of constraint-based problems. It smartly interbreeds ideas from

different domains such as Operations Research, Artificial Intelligence, Graph Theory and Programming Languages. Currently, CP is largely used in different application domains, for instance, in computer graphics, to express geometric coherence, in natural language processing for the construction of efficient parsers, in database systems to ensure and/or restore data consistency, in electrical engineering to locate faults, and even for sequencing the DNA in molecular biology. The principle behind CP is simple: The user states the problem and the system solves it.

In CP, the selection of an enumeration strategy is crucial for the performance of the resolution process, where a correct selection can dramatically reduce the computational cost of finding a solution. However, it is well-known that deciding a priori the correct heuristic is quite difficult, as the effects of the strategy can be unpredictable. In recent years, different efforts have been done to determine good strategies based on the information generated through the resolution process. However, deciding what information must be measured and how to redirect the search is an ongoing research direction.

The enumeration strategies are constituted by variable and value selection heuristics. The idea behind the variable selection heuristic is to select the suitable variable identifying bad situations as rapidly as possible. On the other hand, the goal pursued by the value selection heuristic consists of selecting the value that has more probability of corresponding to a solution. A variable or value selection can be either static, where the selection is fixed and determined prior to search, or dynamic, where the selection criteria is determined as the search progresses. Given a Constraint Satisfaction Problem (CSP) and a tree-based backtracking search algorithm, a variable or value selection is said to be optimal if the selection results in a search that visits the fewest number of nodes over all possible selections when finding one solution or showing that there does not exist a solution. Finding optimal selections is a computationally difficult task. In [13] the author shows that simply deciding whether a variable is the first variable in an optimal variable selection is at least as hard as deciding whether the CSP has a solution. Finding an optimal value selection is also clearly at least as hard since, if a solution exists, an optimal value selection could be used to efficiently find a solution. Thus, such a pair of decisions is crucial in the performance of the resolution process, where a correct selection can dramatically reduce the computational cost of finding a solution. For a simple CSP problem, a good enumeration strategy goes directly to a solution performing a few enumerations without backtracking. However, a bad strategy can perform a lot of backtracks before reaching a solution. Obviously strategies have drastically different efficiencies, often several orders of magnitude, and thus it is crucial to select a good one that unfortunately cannot be predicted in the general case.

There exist various studies about enumeration strategies [7, 2], some centered in defining general criteria, *e.g.*, the smallest domain for variable selection, and its minimum, maximum, or a random value for value selection. As opposed to this idea, some research works have proposed strategies for a given class of problems, *e.g.*, for job shop scheduling [18], as well as for configuration design [8]. We can also find research focused on determining the best strategy based in some criterion [19], *i.e.*, the selection heuristic is determined only once before starting the resolution process,

remaining unchangeable during the whole process. However, deciding a priori the correct heuristics is quite hard, as the effects of the strategy can be unpredictable. In recent years, multiple efforts have been done to determine good strategies based on the information generated through the resolution process.

Considering the aforementioned concerns, we are interested in making good choices for enumeration, *i.e.*, selection of a variable and a value. However, unlike previous research works we focus our research in reacting on the fly, allowing an early replacement of bad-performance strategies without waiting the entire solution process or an exhaustive analysis of a given class of problems. We introduce techniques allowing the identification and measurement of indicators for the resolution process. The main goal is to make possible the classification of the execution process state, considered as the resolution progress, and in that way be able to determine if the current strategy exhibits a poor performance and whether it is necessary to replace it with a better one. Such evaluation procedure is not carried out for improving the resolution of a single problem. We address our approach to efficiently find solutions for different problems. This can be done by exploiting search process features to dynamically adapt a CP solver changing the enumeration strategy in use when an other strategy looks more promising in order to solve the CSP at hand. Regarding this issue, we use Autonomous Search (AS) principles [11] where a system should be able to change its internal components when exposed to changing external forces and opportunities, in order to obtain better results. More precisely, we use a hyperheuristic approach that operates at a higher level of abstraction than the CSP solver. The hyperheuristic has no problem-specific knowledge. It manages a portfolio of enumeration strategies. At any given time the hyperheuristic must choose which enumeration strategy to call. In our approach a choice function adaptively ranks the enumeration strategies and the problem of determining the best set of parameters of the choice function is tackled using a Genetic Algorithm (GA). The contributions of this paper are the following:

- We propose a new algorithm that tackles a critical limitation of common strategies: they remain in use during the whole resolution process with no reaction against a poor search performance. Our algorithm is able to detect inefficiencies and, as a result, replace the enumeration strategy with a better one.
- We perform an indicator-based observation during the solving process. The main purpose of indicators is to proportion the relevant information about the behavior of the resolution process. They must reflect the real state of progress in the problem resolution. In this way, we are able to elaborate a correct judgment about the search performance. To this end, we define simple and quantitative indicators, which can be used different times as well as percentage combinations of them depending on the used search techniques and/or the problem to solve.
- Using AS principles we propose a hyperheuristic approach to decide which enumeration strategy to apply at each decision step during the search. To allow the hyperheuristic to operate, we define a choice function which adaptively ranks the enumeration strategies.

The rest of this paper is organized as follows. In Section 2 we present our approach to meet the challenge of dynamic selecting enumeration strategies for solving CSPs. Section 3 presents the benchmark problems and the experiments settings. We then

present the experiments and the analysis of results. Finally, in Section 4 we conclude and highlight future directions of research.

2. Dynamic Selection of Enumeration Strategies using an Hyperheuristic Approach

The research community in constraint solving is fruitful in heuristics to solve CSPs, but the efficiency of a heuristic may vary dramatically depending on the problem at hand. A combination of heuristics seems to be a reasonable approach, but that combination must be established [17].

A CSP can be solved with a search algorithm that specifies heuristics for variable and value selection using inference and backtracking methods. There are a broad range of variable ordering and value ordering heuristics that try to improve the search. However, no single heuristic is the best for all CSP classes [22]. This work therefore seeks a combination of heuristics, betting that a good combination of heuristics can outperform even the best individual heuristic.

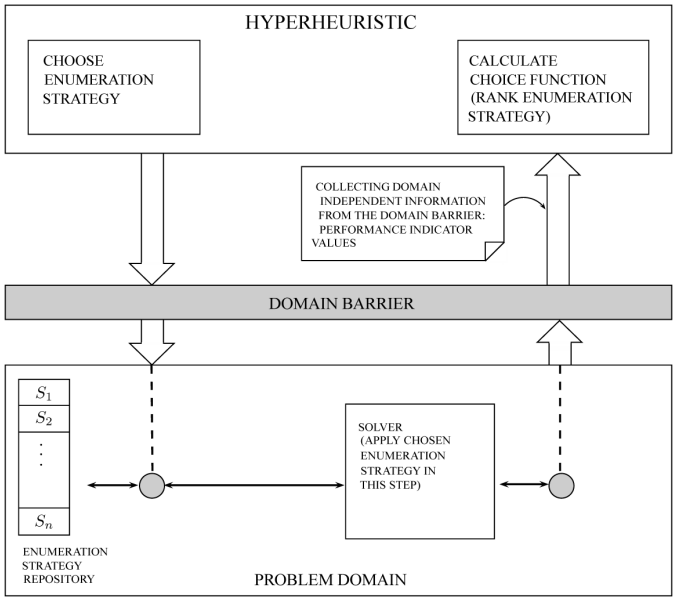

Figure 1. Hyperheuristic framework based on a choice function.

On the one hand, an *a priori* decision concerning a good combination of enumeration strategies (variable + value selection heuristics) is very difficult. Heuristics that sample information before or during search in order to inform subsequent decisions have shown better performance and greater robustness than fixed heuristics [5].

In this section we present a Hyperheuristic approach which decides which Enumeration Strategy to apply at each decision step during the search. Then we present the details of the proposal including the indicators used to measure the solving process,

a Choice Function that determines the performance of a given Enumeration Strategy in a given amount of time through a set of indicators and control parameters, and the data flow during the execution of the Hyperheuristic approach proposed. Finally we present a GA used to tune the parameters of the choice function. From the point of view of software architecture, our approach is supported by the architecture proposed in [16] and developed in [10].

2.1. The Hyperheuristic Approach

Hyperheuristic systems employ one of two approaches in deciding on which constructive heuristic to apply next. The first either identifies or adapts an existing heuristic to be applied at each stage of the construction process, while the second approach optimises a search space of heuristic combinations, *i.e.*, list of low-level heuristics. The study presented in this work focuses on the second approach taken by Hyperheuristic systems, *i.e.*, the generation of combinations of low-level heuristics.

A hyperheuristic approach is a heuristic that operates at a higher level of abstraction than the CSP solver [6]. The hyperheuristic has no problem specific knowledge. At any given time the hyperheuristic must choose which Enumeration Strategy to call. To allow the Hyperheuristic to operate, we define a Choice Function which adaptively ranks the Enumeration Strategies [9].

The Choice Function provides guidance to the Hyperheuristic by indicating which Enumeration Strategy should be applied next based upon the information of the search process (it should be captured through some indicators). The Choice Function is defined as a weighted sum of indicators (a linear combination) expressing the recent improvement produced by the Enumeration Strategy called.

2.1.1. Measuring the Solving Process: Indicators

The proposed adaptive model must possess the ability to change a strategy to another according to the effect that these strategies have in the resolution process, *i.e.*, to change the strategy having a bad performance, assuming that other strategy might work better. To measure the effect of a strategy in the resolution process and make the decision of changing or not, it must be a process supported by the observation of information generated during the resolution process.

An indicator provides information that will enable us to know the state of progress in the solution process for a problem. Indicators can be divided into *base indicators* and *calculated indicators* (from the base indicators), see Table 1 and 2 respectively.

2.1.2. Indicators for the Search-tree Process Cost

The cost measures allow us to decide when one strategy performs better than another [3]. While the comparison with other techniques may be done once the resolution of the problem has ended, some of these indicators could be used in intermediate stages of the process to assess the performance of resolution process at a specific time. Example of this kind of indicators are:

- Number of backtracks (Backtracks) [3, 4, 16, 19]: Counts the number of times

the resolution process goes back from a variable x_j to its predecessor x_i after having proved that none of the extensions of x_j can be extended to a solution. In terms of the search tree, it counts the number of times the search goes up in the tree from a node u to its predecessor after having exhausted the last child of u. The condition for a node u to be counted as a backtrack node is that it has some children which have been visited, and that none of them was successful.

- Number of nodes (Nodes) [3]: Counts the number of nodes visited.
- Runtime/CPU time [16, 4]: Measures the time required in solution process of a problem.

Table 1. Base Indicators

Code	**Name (Base Indicators)**	**Description**	**Calculate**	**References**
Step	Number of Measurements	Every time the choice function is used	Step ++	
TV	Total Number of Variables	Total Number of Variables in the Problem. It is an indicator in relation with the characteristic of the problem, rather than the solving process. Its value is constant during the process	length(AllVars,N)	
VU	Variables Unfixed	Number of Variables Unfixed	var_count(Rest, VU). Conceptually corresponds to length(Rest,N) - Vt1. Where "Vt1" is the number of variables with domain size equal to 1 in the Variables Unfixed set. Note that "Vt1" is equal to VFPS (see Table 2).	
SB	Shallow Backtracks	When we try to assign a value to the current variable, the constraint propagation process is automatically triggered. If it yields a failure, the next value is tried.		[1]
SS	Search Space	Current Search Space	$\prod(Dom_i)_t$	[16]
SS_pr	Previous Search Space	Previous Search Space	$\prod(Dom_i)_{t-1}$	[16]
B	Backtracks	If the Current Variable causes a Dead-End (Propagation that produces an Empty Domain) then the Algorithm Backtracks to the Previously Instantiated Variable	Every Time you Try to Perform a B. A counter is set to increase by one	[3], [12], [14], [4], [16], [19]
N	Nodes	Number of Nodes Visited	A counter is set to increase by one every time you visit a node	[3]

Table 2. Calculated Indicators (to be continued on next page)

Code	Name (Calculated Indicators)	Description	Calculate	References
VF	Variables Fixed	Number of Variables Fixed by Enumeration and Propagation	$(TV - VU)$	[12, 7]
VFES	Variables Fixed by Enumeration in each Step	Number of Variables Fixed by Enumeration in each Step. It makes sense when the number of steps is greater than 1	1	[16], [7]
TAVFES	Total Accumulated of Variables Fixed by Enumeration in each Step	Number of Total Accumulated Variables Fixed by Enumeration in each Step	$\sum(VFES)$	[16], [7]
VFPS	Variables Fixed by Propagation in each Step	Number of Variables Fixed by Propagation in each Step. Note that VFPS is equal to "Vt1" (see Table 1)	$length(Rest, N) - VU$	[16]
TAVFPS	Total Accumulated of Variables Fixed by Propagation in each Step	Number of Total Accumulated Variables Fixed by Propagation in each Step	$\sum(VFPS)$	[16]
TSB	Total Shallow Backtracks	Accumulated Number of Shallow Backtracks	$\sum(SB)$	[1]
d_pr	Previous Depth	Previous Depth of the current node in the Search Tree		[16]
d	Depth	Current Depth of the current node in the Search Tree		[16], [7]
dmax	Maximum Depth	Current Maximum Depth Reached in the Search Tree	$dmax$ is d, if $d \geq dmax\_pr$	[16]
dmax_pr	Previous Maximum Depth	Previous Maximum Depth Reached in the Search Tree	$MAX(dmax)_{n-1}$	[16]
In1	Current Maximum Depth - Previous Maximum Depth	Represents a Variation of the Maximum Depth.	$(dmax - dmax\_pr)$	[16]
In2	Current Depth - Previous Depth	Positive means, that the Current Node is Deeper than the one Explored at the Previous Step.	$(d_t - d_{t-1})$	[16]
In3	Reduction Search Space	If Positive, The Current Search Space is Smaller than the one at Previous Snapshot.	$(((SS_{t-1} - SS_t)/SS_{t-1}) * 100)$	[16]
PVFES	Percentage of Variables Fixed by Enumeration in each Step	It makes sense when the number of steps is greater than 1	$((VFES/TV) * 100)$	[7]

Table 2. Calculated Indicators (continued from previous page)

Code	**Name (Calculated Indicators)**	**Description**	**Calculate**	**References**
PTAVFES	Percentage of Total Accumulated of Variables Fixed by Enumeration in each Step		$((TAVFES/TV)*100)$	[7]
PVFPS	Percentage of Variables Fixed by Propagation in each Step		$((VFPS/TV) * 100)$	[7]
PTAVFPS	Percentage of Total Number of Variables Fixed by Propagation in each Step		$((PTAVFPS/TV) * 100)$	[7]
Thrash	Thrashing	The Solving Process alternates Enumerations and Backtracks on a few Number of Variables without Succeeding in having a Strong Orientation.	$(d_{t-1} - VFPS_{t-1})$	[7]
DB	Depth Back	If a backtrack is performed. Count how many levels backward in the Depth	If $D - d_pr = 0$, then add 1. If $D < d_pr$, then increases in $(d_pr - D) + 1$. In another case maintain constant	[7]

2.1.3. Correlation Analysis

Collecting the indicators and using all of them in the choice function is a costly task (it is a combinatorial problem in itself). We thus use correlation analysis to detect indicators that are not useful because they behave as an other indicator. A correlation analysis was performed to detect pairs or sets of highly related indicators, and thus equivalent. We use the Pearson Correlation Coefficient, which is sensitive to a linear relationship between two variables. It is obtained by dividing the Covariance of two variables by the product of their Standard Deviations

$$\rho_{X,Y} = corr(X,Y) = COV(X,Y)/\sigma_X\sigma_Y \tag{1}$$

The Pearson Correlation is defined only if both of the standard deviations are finite and both of them are nonzero. The Correlation Coefficient is symmetric (then $corr(X,Y) = corr(Y,X)$) and scale free. The Pearson Correlation is 1 in the case of a perfect positive linear correlation, –1 in the case of a perfect negative linear relationship (anticorrelation). Values between –1 and 1 indicate the degree of linear dependence between the variables. When it is near to zero, there is less correlation.The Correlation Matrix of n variables $X_1, ..., X_n$ is the $n \times n$ matrix whose i, j entry is $corr(X_i, X_j)$.

The criteria used to detect pairs of indicators highly related was an absolute value of correlation coefficient at least of 0.9. We identified the indicators highly related (see Tables 3a and 3b), then we will use only one of them (one per row) in our Choice Functions. Thus, for example, we will use *VF* (the number of fixed variables) and not *d* (the current depth in the search tree) since they are highly related, and thus redundant.

Table 3a. Indicators highly related (group by row)

	Positive Linear Relationship					
1	B	N	TAVFES	TSB	PTAVFES	DB
2	VF	d				
3	VFPS	PVFPS				
4	d_pr	Thrash				

Table 3b. Indicators highly related (group by row)

	Negative Linear Relationship	
1	VU	VF
2	VU	d

2.1.4. Choice Function

As previously mentioned, the CSP search phase is commonly tackled by building a tree structure by interleaving enumeration and propagation phases. In the enumeration phase, the branches of the tree are created by selecting variables and values from their domains. In the propagation phase, a consistency level is enforced to prune the tree in order to avoid useless tree inspections.

Procedure 1 (see Figure 2a) represents a general procedure for solving CSPs. The goal is to recursively generate partial solutions, backtracking when an inconsistency is detected, until a result is reached. The algorithm uses two data structures: *inst* and *D*. The former holds the instantiations while the latter holds the set of domains. The variable *k* represents the current level of the tree and *success* is a Boolean variable to be set to true when a solution is found. The *instantiate* function is responsible for building the partial solutions and assigning them into the *inst* array. The *consistent* function decides whether the current instantiation can be extended to a full solution; additionally, it sets *success* to true if the current instantiation is a solution. At the end, *restore* reinitializes the domain of the *k* variable.

Let us notice that the value and variable selections are performed at line 2 and 10, respectively. The *propagate* procedure is responsible for pruning the tree by enforcing a consistency property on the constraints of the problem. The most used notion of consistency is the arc-consistency [21].

Procedure 1 $solve(k : \textbf{integer}, inst : \textbf{array})$

```
1: while D[k] ≠ {} and not success do
2:   a ← choose_value_from(D[k])
3:   inst ← instantiate(inst, k, a)
4:   if consistent(inst, k, success) then
5:     if success then
6:       print solution(inst)
7:     else
8:       propagate(k, D, failure)
9:       if not failure then
10:        l ← choose_variable()
11:        solve(l, inst)
12:      end if
13:    end if
14:  end if
15: end while
16: restore(k);
```

(a) General solving procedure.

Procedure 2 $solve(k : \textbf{integer}, inst : \textbf{array})$

```
1: while D[k] ≠ {} and not success do
2:   a ← choose_value_from2(D[k])
3:   inst ← instantiate(inst, k, a)
4:   calculate_indicators()
5:   calculate_choice_functions()
6:   choose_promising_enum_strategy()
7:   if consistent(inst, k, success) then
8:     if success then
9:       print solution(inst)
10:    else
11:      propagate(k, D, failure)
12:      if not failure then
13:        l ← choose_variable2()
14:        solve(l, inst)
15:      end if
16:    end if
17:  end if
18: end while
19: restore(k);
```

(b) General solving procedure including the choice function.

Figure 2. Procedures for solving CSPs.

As mentioned above, our hyperheuristic is based upon a choice function, which adaptively ranks the enumeration strategies. The choice function value of each enumeration strategy is determined based on information with regards to performance

indicators. The choice function attempts to capture the correspondence between the historical performance of each enumeration strategy and the decision point currently being investigated. Here, a decision point or step is performed every time the solver is invoked to fix a variable by enumeration.

The choice function is used to rank and choose between different enumeration strategies at each step. For any enumeration strategy S_j, the choice function f in step n for S_j is defined by Equation 2, where l is the number of indicators considered and α is a parameter to control the relevance of the indicator within the choice function.

$$f_n(S_j) = \sum_{i=1}^{l} \alpha_i f_{i_n}(S_j) \tag{2}$$

Additionally, to control the relevance of an indicator i for a strategy S_j in a period of time, we use a popular statistical technique for producing smoothed time series called exponential smoothing. The idea is to associate greater importance to recent performance by exponentially decreasing weights to older observations. The exponential smoothing is applied to the computation of $f_{i_n}(S_j)$, which is defined by equations 3 and 4, where v_{i_1} is the value of the indicator i for the strategy S_j in time 1, n is a given step of the process, β is the smoothing factor, and $0 < \beta < 1$.

$$f_{i_1}(S_j) = v_{i_1} \tag{3}$$

$$f_{i_n}(S_j) = v_{i_n} + \beta_i f_{i_{n-1}}(S_j) \tag{4}$$

Then the exponentially smoothed moving average for step n is given by

$$f_{i_n}(S_j) = v_{i_n} + \beta_i v_{i_{n-1}} + \beta_i^2 v_{i_{n-2}} + \beta_i^3 v_{i_{n-3}} + \text{.....} \tag{5}$$

Let us note that the speed at which the older observations are smoothed (dampened) depends on β. When β is close to 0, dampening is quick, and when it is close to 1, dampening is slow.

The general solving procedure including the choice function can be seen in Fig. 2b. Three new function calls have been included: for calculating the indicators (line 4), the choice functions (line 5), and for choosing promising strategy (line 6), that is, the one with highest choice function. They are called every time the solver is invoked to fix a variable by enumeration, *i.e.*, after instantiation. Let us notice that procedures for selecting variables and values (lines 2 and 13) have been modified to respond to the dynamic replacement of strategies.

2.1.5. Choice Function Tuning with a Multilevel Structure

The search for the best tuning of parameters α_i of the CSP solver (based on the Choice Function) may be formulated as an optimization problem. Hence, this meta-optimization approach may be performed by a metaheuristic. This approach is composed of two levels [20]: the meta-level and the base level. At the meta-level, a metaheuristic operates on solutions representing the parameters of the metaheuristic

to optimize. A solution X at the meta-level will represent all the parameters we want to optimize. At the meta-level, the objective function f_m associated with a solution X is a performance indicator of the metaheuristic using the parameters specified by the solution X. Hence, to each solution X of the meta-level will correspond an independent metaheuristic in the base level. The metaheuristic of the base level operates on solutions that encode solutions of the original problem. The objective function f_b used by the metaheuristics of the base level is associated with the target problem. The study presented in this work at the meta level employs a Genetic Algorithm to search the heuristic space of choice functions of the base level (see Fig. 3).

Then, in order to determine the most appropriate set of parameters α_i for the choice function we use a multilevel approach. The parameters are fine-tuned by a GA which trains the choice function carrying out a sampling phase. Sampling occurs during an initial information gathering phase where the search is run repeatedly until a fix cutoff (*i.e.*, until a fixed number of variables instantiated, number of visited nodes, number of backtracks or number of steps). After sampling, the problem is solved with the most promising set of parameter values for the choice function.

The upper-level GA simulates, evaluates and evolves different combinations of parameters, relieving the task of manual parameterization. Each member of the population encodes the parameters of a choice function, then these individuals are used in order to create a choice function instance. Each choice function instantiated (*i.e.* each chromosome) is evaluated in a sampling phase trying to solve partially the problem at hand. An indicator of process performance is used as a fitness value for the chromosome, (number of visited nodes, number of backtracks or number of steps). After each chromosome of the population is evaluated, selection, crossover and mutation are used to breed a new population of choice functions. As noted above, the multilevel approach is used to tune the choice function, the resulting choice function is applied to solve the entire CSP problem.

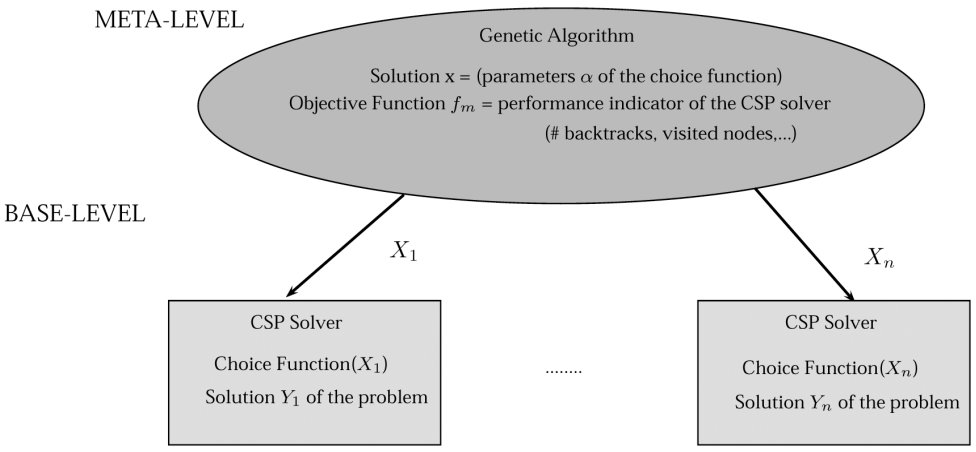

Figure 3. Multilevel parameterization.

2.1.6. Workflow through the Hyperheuristic Approach

In this section, we provide a description of the workflow during the execution of the hyperheuristic approach proposed in this work. We describe the main steps of the hyperheuristic approach and then we detail the entire solving process.

From a high level point of view, our approach works as follows. The first step fixes the solving options; a portfolio of enumeration strategies and fixes the indicators (see Section 2.1.1) of the choice function. The evaluation value of the GA fitness (*i.e.*, number of visited nodes, number of backtracks or number of steps) and the cutoff value (*i.e.* percentage or number of fixed variables or number of visited nodes or number of backtracks or number of steps) of process for tuning the choice function are fixed. Then, the choice function is tuned and the problem solving starts.

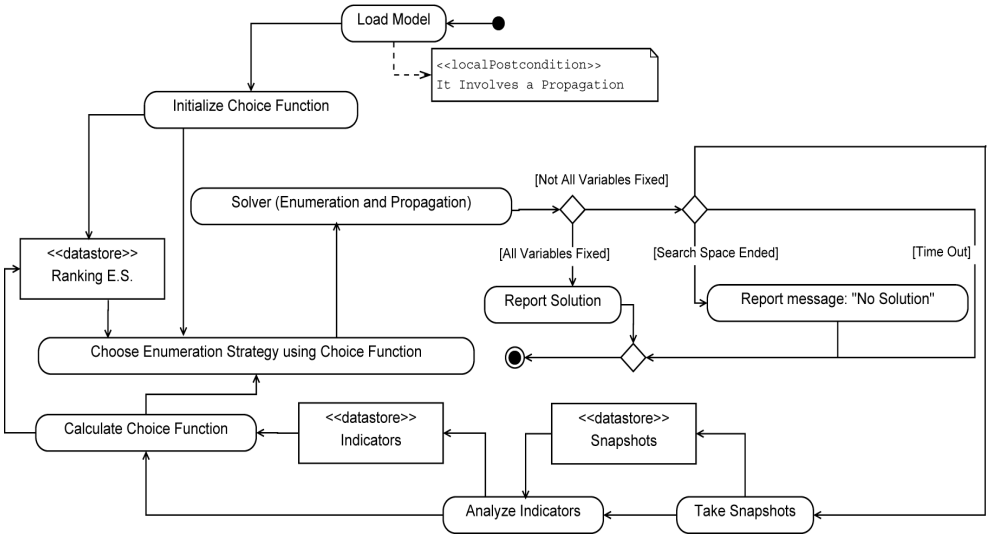

Figure 4. Problem Solving activity diagram.

The problem solving (see Fig. 4) starts loading a model where the variables of the problem, the domains of them, and the constraints are defined. Finally, a constraint propagation process starts in order to detect local inconsistencies earlier and prune the domains. Once a model has been loaded, the choice functions are zero initialized (for all the enumeration strategies) for the purpose of choosing an enumeration strategy to apply by the solver. Then the system takes snapshots and the snapshots feed the process of indicator analysis. Subsequently, the choice function is evaluated, the enumeration strategies are ranked and an enumeration strategy is selected continuing the cycle of the solving process. Finally, if all the variables have been fixed, a solution is reported. The choice function breaks ties randomly [1].

[1]It is handled in a way keeping a "good previous random ordering", that is, an ordering that has been used in the choice function tuning phase to break ties.

3. Experimental Evaluation

In this Section, we provide a performance evaluation of our approach. We have implemented a solver using ECL^{i}PS^e Constraint Programming System version 5.10 and java with NetBeans IDE 6.7.1. Tests have been performed on a 2.33GHZ Intel Core2 Duo with 2GB RAM running Windows XP. The problems used to test our approach were the following: N-queens (NQ), 10 linear equations (eq-10), 20 linear equations (eq-20), Magic square (MS), Latin square (LS), Sudoku (Sud) and Knight tour problem (KTP).

The variable selection heuristics used in the experiments are:

- *input_order*: the first entry in the list is selected.
- *anti_first_fail*: the entry with the largest domain size is selected.
- *first_fail*: the entry with the smallest domain size is selected (also known as minimun remaining values).
- *occurrence*: the entry with the largest number of attached constraints is selected.

The value selection heuristics used are:

- *indomain(Var,min)*: it starts with the smallest element and upon backtracking tries successive elements until the entire domain has been explored.
- *indomain(Var,max)*: it starts the enumeration from the largest value downwards.

The combination of variable and value selection heuristics generates eight enumeration strategies as shown in Table 4.

Table 4. Enumeration strategies used

S_1 = input_order+indomain(Var,min)
S_2 = anti_first_fail+indomain(Var,min)
S_3 = first_fail+indomain(Var,min)
S_4 = occurrence+indomain(Var,min)
S_5 = input_order+indomain(Var,max)
S_6 = anti_first_fail+indomain(Var,max)
S_7 = first_fail+indomain(Var,max)
S_8 = occurrence+indomain(Var,max)

Considering the correlation analysis of the indicators we considered the following combinations of them in order to test some different choice functions (see Table 5).

Table 5. Choice functions used in the experiments

CF_1: $\alpha_1 B + \alpha_2 VF + \alpha_3 VFPS + \alpha_4 Thrash$	CF_{13}: $\alpha_1 N + \alpha_2 d + \alpha_3 VFPS + \alpha_4 Thrash$
CF_2: $\alpha_1 B + \alpha_2 VF + \alpha_3 VFPS + \alpha_4 d\_pr$	CF_{14}: $\alpha_1 N + \alpha_2 d + \alpha_3 VFPS + \alpha_4 d\_pr$
CF_3: $\alpha_1 B + \alpha_2 VF + \alpha_3 PVFPS + \alpha_4 Thrash$	CF_{15}: $\alpha_1 N + \alpha_2 d + \alpha_3 PVFPS + \alpha_4 Thrash$
CF_4: $\alpha_1 B + \alpha_2 VF + \alpha_3 PVFPS + \alpha_4 d\_pr$	CF_{16}: $\alpha_1 N + \alpha_2 d + \alpha_3 PVFPS + \alpha_4 d\_pr$
CF_5: $\alpha_1 B + \alpha_2 d + \alpha_3 VFPS + \alpha_4 Thrash$	CF_{17}: $\alpha_1 B + \alpha_2 VU + \alpha_3 VFPS + \alpha_4 Thrash$
CF_6: $\alpha_1 B + \alpha_2 d + \alpha_3 VFPS + \alpha_4 d\_pr$	CF_{18}: $\alpha_1 B + \alpha_2 VU + \alpha_3 VFPS + \alpha_4 d\_pr$
CF_7: $\alpha_1 B + \alpha_2 d + \alpha_3 PVFPS + \alpha_4 Thrash$	CF_{19}: $\alpha_1 B + \alpha_2 VU + \alpha_3 PVFPS + \alpha_4 Thrash$
CF_8: $\alpha_1 B + \alpha_2 d + \alpha_3 PVFPS + \alpha_4 d\_pr$	CF_{20}: $\alpha_1 B + \alpha_2 VU + \alpha_3 PVFPS + \alpha_4 d\_pr$

The genetic algorithm used to obtain the best weights (α's) of the choice functions was implemented using the following parameters: number of generation=20, population size=30,crossover type=uniform, crossover rate=0.4, mask probability=0.5, mutation rate=0.025, selector tournament size=3, tournament selector parameter=0.75, fitness=number of visited nodes. The GA was implemented using the Java Genetic Algorithm Package[2] (JGAP) version 3.5. The basic behavior of a GA implemented using JGAP contains three main steps: a setup phase, the creation of the initial population and the evolution of the population.

For comparison with our choice function approach (using the portfolio of 8 enumeration strategies), we considered 8 enumeration strategies used solely (F+ID, AMRV + ID, MRV + ID, O + ID, F + IDM, AMRV + IDM, MRV + IDM, and O + IDM), and a random selection. In the experiments with Random and Choice Function approaches we show **min** (the minimum value), **avg** (the average obtained of the experiments where a solution was found before the timeout of 65535 steps), **sd** (the standard deviation), **max** (the maximum value) and [%] (the percentage of the experiments performed before the timeout is reached).

3.1. Results

For each measurement, the results were obtained over 30 experiments. Due to space constraints, in this work we present results based on number of backtracks. Tables 6a, 6b, 7a, 7b and 7c present the results measured in terms of number of backtracks. More precisely, Table 6a shows the results related to the N-Queens problem. Tables 6b, 7a, 7b and 7c show the results related to the problems Magic square, Sudoku, Latin square and Knight tour respectively.

3.2. Discussion of Results

Our results agree with previous works showing good performance for strategies following the first-fail principle for variable selection (starting with those variables more difficult to assign values), strategies S_3 and S_7 in our work. Our decision-making proposal is a structured process and the research shows that its results are, on average, clearly better than the random selection of enumeration strategies. For many problems one of our choice functions was at least in top-3 ranking, finding effective dynamic ways to improve the commutation of enumeration strategies during the search. This show the ability of our approach to adapt itself to a different problem and that it could adapt itself and converge to an efficient strategy for the problem at hand. In agreement with other algorithms devised for constraint satisfaction that depend on sampling before or during the search, our approach was very encouraging. The sampling, performed in the choice function tuning phase, is a kind of random probing [23] where the global improvements are due in large part to better variable selections at the top of the search tree. After observing the trace, we have verified that if a good strategy from the portfolio is selected at the top of search tree, it is maintained operating. In other case, a badly evaluated strategy is replaced quickly.

[2] http://jgap.sourceforge.net/

Table 6a. Number of Backtracks solving N-Queens problem with different strategies

Strategy	NQ n=8	NQ n=9	NQ n=10	NQ n=11	NQ n=12
S_1	10	2	6	2	15
S_2	11	2	12	39	11
S_3	10	2	4	5	16
S_4	10	2	6	2	15
S_5	10	2	6	2	15
S_6	11	2	12	39	11
S_7	10	2	4	5	16
S_8	10	2	6	2	15
	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]
Random	(4) 7,5 ± 2,2 (12) [100]	(1) 5,3 ± 3,6 (11) [100]	(2) 9,5 ± 5,3 (21) [100]	(0) 15,5 ± 8,6 (33) [100]	(2) 16,4 ± 11,7 (39) [100]
CF_1	(4) 5,7 ± 2,2 (10) [100]	(1) 4,5 ± 3,6 (11) [100]	(4) 7,1 ± 2 (11) [100]	(0) 5,1 ± 6,4 (21) [100]	(1) 14,9 ± 9,8 (48) [100]
CF_2	(4) 6,3 ± 2,5 (10) [100]	(1) 3,8 ± 3,4 (10) [100]	(4) 6,4 ± 1,5 (9) [100]	(0) 5,1 ± 8,3 (38) [100]	(0) 16,5 ± 13,4 (46) [100]
CF_3	(4) 6,4 ± 2,2 (10) [100]	(1) 4 ± 3,2 (10) [100]	(2) 6,4 ± 1,9 (10) [100]	(0) 5,9 ± 7,5 (27) [100]	(0) 11,5 ± 10,4 (48) [100]
CF_4	(4) 6,4 ± 2,2 (10) [100]	(1) 4,4 ± 3 (10) [100]	(4) 7 ± 1,6 (10) [100]	(0) 4 ± 5,1 (20) [100]	(0) 13,8 ± 7,9 (25) [100]
CF_5	(4) 5,6 ± 2,2 (10) [100]	(1) 4,2 ± 3,5 (10) [100]	(4) 6,5 ± 1,5 (10) [100]	(0) 4,7 ± 7,8 (34) [100]	(0) 14,2 ± 10,6 (47) [100]
CF_6	(4) 5,8 ± 2,1 (10) [100]	(1) 4,8 ± 3,9 (10) [100]	(4) 6,4 ± 1,4 (11) [100]	(0) 4,1 ± 5 (16) [100]	(2) 13,9 ± 8 (30) [100]
CF_7	(4) 6,2 ± 2,2 (10) [100]	(1) 3,9 ± 3,2 (10) [100]	(5) 6,6 ± 1,3 (10) [100]	(0) 4,4 ± 6,8 (35) [100]	(0) 12 ± 8,8 (29) [100]
CF_8	(4) 5,7 ± 2 (10) [100]	(1) 4,4 ± 3,7 (10) [100]	(4) 6,8 ± 1,4 (10) [100]	(0) 6,5 ± 9,9 (37) [100]	(3) 16,7 ± 13 (50) [100]
CF_9	(4) 6 ± 2 (10) [100]	(1) 4,5 ± 3,5 (11) [100]	(5) 7,3 ± 2 (12) [100]	(0) 5,2 ± 6,2 (19) [100]	(0) 9,9 ± 8,8 (31) [100]
CF_{10}	(4) 5,1 ± 1,7 (10) [100]	(1) 4,3 ± 3,5 (10) [100]	(4) 6,6 ± 1,6 (12) [100]	(0) 5,4 ± 7,5 (31) [100]	(1) 13,6 ± 9,7 (45) [100]
CF_{11}	(4) 6,2 ± 2,3 (10) [100]	(1) 4,2 ± 3,3 (10) [100]	(4) 6,7 ± 1,2 (10) [100]	(0) 6,6 ± 9 (32) [100]	(2) 12,4 ± 8,4 (39) [100]
CF_{12}	(4) 6,3 ± 2,1 (10) [100]	(1) 4,2 ± 3,4 (11) [100]	(4) 6,9 ± 1,9 (12) [100]	(0) 3,2 ± 4,9 (20) [100]	(0) 14,2 ± 10,8 (45) [100]
CF_{13}	(4) 6,2 ± 2,2 (10) [100]	(1) 4,5 ± 3,6 (11) [100]	(4) 6,6 ± 1,6 (10) [100]	(0) 4,9 ± 6 (17) [100]	(0) 15,2 ± 10,3 (44) [100]
CF_{14}	(4) 5,7 ± 2,1 (10) [100]	(1) 5,1 ± 3,7 (11) [100]	(5) 7,2 ± 1,4 (11) [100]	(0) 6,7 ± 8,7 (34) [100]	(0) 13,5 ± 10,8 (47) [100]
CF_{15}	(4) 5,9 ± 1,9 (10) [100]	(1) 4,3 ± 3,6 (10) [100]	(4) 6,9 ± 1,6 (11) [100]	(0) 4,7 ± 6,1 (21) [100]	(0) 9,5 ± 6,6 (21) [100]
CF_{16}	(4) 6,5 ± 2,3 (10) [100]	(1) 4,3 ± 3,4 (10) [100]	(4) 6,8 ± 1,6 (10) [100]	(0) 2,9 ± 4,4 (16) [100]	(1) 13,8 ± 7,9 (30) [100]
CF_{17}	(4) 6,8 ± 2,4 (10) [100]	(1) 4,1 ± 3,6 (10) [100]	(2) 6,4 ± 2,1 (13) [100]	(0) 5,4 ± 8,5 (41) [100]	(1) 15,5 ± 9,9 (37) [100]
CF_{18}	(4) 6,7 ± 1,9 (10) [100]	(1) 3,5 ± 3,4 (11) [100]	(4) 6,5 ± 1,7 (11) [100]	(0) 8 ± 8,9 (32) [100]	(1) 13,9 ± 12,5 (45) [100]
CF_{19}	(4) 6,4 ± 1,9 (10) [100]	(1) 3,3 ± 3 (11) [100]	(4) 6,9 ± 2,4 (16) [100]	(1) 6,2 ± 7,8 (34) [100]	(1) 12,7 ± 10 (47) [100]
CF_{20}	(4) 7,1 ± 2,5 (10) [100]	(1) 5,1 ± 3,8 (11) [100]	(2) 6,2 ± 2 (10) [100]	(0) 7,9 ± 9,6 (35) [100]	(0) 15,8 ± 10,1 (50) [100]
CF_{21}	(4) 5,9 ± 2,1 (10) [100]	(1) 4,5 ± 3,4 (10) [100]	(4) 6,8 ± 1,7 (11) [100]	(1) 6,3 ± 10,1 (35) [100]	(0) 15,4 ± 11,3 (45) [100]
CF_{22}	(4) 6 ± 2,1 (10) [100]	(1) 3,9 ± 3,5 (10) [100]	(4) 6,7 ± 1,4 (10) [100]	(0) 5,1 ± 6,7 (31) [100]	(0) 14 ± 8,1 (27) [100]
CF_{23}	(4) 6,4 ± 2,4 (10) [100]	(1) 3,6 ± 3,2 (10) [100]	(4) 6,7 ± 2,1 (13) [100]	(0) 2,8 ± 4,1 (16) [100]	(1) 14,6 ± 7,5 (36) [100]
CF_{24}	(4) 6 ± 2 (10) [100]	(1) 4,5 ± 3,6 (11) [100]	(2) 6,7 ± 2,6 (11) [100]	(0) 5,1 ± 6,6 (28) [100]	(0) 14,6 ± 11,5 (50) [100]

Table 6b. Number of Backtracks solving Magic Square problem with different strategies

Strategy	MS n=3	MS n=4	MS n=5
S_1	0	12	910
S_2	4	1191	>46675
S_3	0	3	185
S_4	0	10	5231
S_5	1	51	>47748
S_6	0	42	>44157
S_7	1	97	>47935
S_8	1	29	>39008
	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]
Random	(0) 0,7 ± 1,2 (4) [100]	(0) 31,1 ± 43,9 (184) [100]	(1) 3831 ± 8402,7 (37986) [66,7]
CF_1	(0) 0,7 ± 1 (4) [100]	(0) 51,8 ± 67,3 (181) [100]	(0) 1615,5 ± 4447,1 (19640) [70]
CF_2	(0) 0,5 ± 0,8 (4) [100]	(0) 51,9 ± 74,8 (182) [100]	(0) 976,9 ± 3246,3 (13113) [53,3]
CF_3	(0) 0,7 ± 1,2 (4) [100]	(0) 38,3 ± 50,3 (181) [100]	(0) 1502,5 ± 4562,7 (19633) [80]
CF_4	(0) 1,1 ± 1,5 (4) [100]	(0) 38,4 ± 50,8 (181) [100]	(0) 734,9 ± 2683,9 (12906) [76,7]
CF_5	(0) 0,5 ± 0,8 (4) [100]	(0) 45,3 ± 62,4 (181) [100]	(0) 892,6 ± 1689,4 (6346) [80]
CF_6	(0) 0,8 ± 1 (4) [100]	(0) 54,6 ± 58,2 (193) [100]	(0) 586,4 ± 1424,8 (6656) [83,3]
CF_7	(0) 0,8 ± 1,2 (4) [100]	(0) 69,9 ± 74,4 (181) [100]	(3) 1716,4 ± 3276,2 (12918) [76,7]
CF_8	(0) 0,8 ± 1,2 (4) [100]	(0) 45,2 ± 65,4 (181) [100]	(0) 219,9 ± 346,4 (1441) [70]
CF_9	(0) 0,9 ± 1,5 (4) [100]	(0) 35,7 ± 46,8 (149) [100]	(0) 683,1 ± 2464 (12906) [90]
CF_{10}	(0) 0,5 ± 0,8 (4) [100]	(0) 43,9 ± 52,8 (181) [100]	(0) 1047,6 ± 2846,9 (12906) [76,7]
CF_{11}	(0) 0,9 ± 1,3 (4) [100]	(0) 49,3 ± 62,2 (181) [100]	(4) 1233,3 ± 3421,4 (14823) [63,3]
CF_{12}	(0) 0,4 ± 0,8 (4) [100]	(0) 34,3 ± 52,6 (181) [100]	(0) 2253,5 ± 5095,7 (19632) [70]
CF_{13}	(0) 1,2 ± 1,6 (4) [100]	(0) 51,3 ± 59,7 (181) [100]	(15) 1149,2 ± 3289,8 (14823) [70]
CF_{14}	(0) 0,8 ± 1,2 (4) [100]	(0) 25,3 ± 33,5 (111) [100]	(0) 993,4 ± 2015,2 (6988) [56,7]
CF_{15}	(0) 1,1 ± 1,4 (4) [100]	(0) 40,1 ± 53,7 (174) [100]	(0) 1435,5 ± 3721,6 (14823) [53,3]
CF_{16}	(0) 0,9 ± 1,3 (4) [100]	(0) 60,1 ± 57,2 (181) [100]	(0) 1007,2 ± 3280 (14827) [66,7]
CF_{17}	(0) 0,5 ± 0,8 (4) [100]	(2) 56,4 ± 69,1 (181) [100]	(11) 729 ± 1028 (3138) [53,3]
CF_{18}	(0) 1 ± 1,3 (4) [100]	(0) 35,1 ± 54,4 (179) [100]	(1) 2962,9 ± 8011,8 (33285) [66,7]
CF_{19}	(0) 1 ± 1,3 (4) [100]	(0) 36 ± 52,4 (181) [100]	(3) 5152,6 ± 8489,5 (23598) [50]
CF_{20}	(0) 0,7 ± 1,2 (4) [100]	(1) 35,4 ± 48,6 (160) [100]	(1) 753,6 ± 1206,1 (4576) [56,7]
CF_{21}	(0) 1,1 ± 1,6 (4) [100]	(0) 40 ± 62,8 (181) [100]	(0) 582,8 ± 989,6 (3638) [46,7]
CF_{22}	(0) 0,9 ± 1,2 (4) [100]	(0) 37,5 ± 59,6 (181) [100]	(0) 397,6 ± 912,5 (4061) [70]
CF_{23}	(0) 0,8 ± 1,4 (4) [100]	(0) 60,8 ± 76,1 (181) [100]	(0) 652,8 ± 969,2 (3208) [60]
CF_{24}	(0) 0,7 ± 1 (4) [100]	(0) 33,8 ± 52,9 (181) [100]	(18) 913,3 ± 1655,8 (6933) [66,7]

Table 7a. Number of Backtracks solving Sudoku problem with different strategies

Strategy	Sud 2	Sud 12
S_1	18	0
S_2	10439	1
S_3	4	2
S_4	18	0
S_5	2	3
S_6	6541	3
S_7	9	1
S_8	2	3
	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]
Random	(3) 101,2 ± 71,9 (261) [100]	(0) 1,8 ± 1,2 (4) [100]
CF_1	(0) 5,5 ± 5,7 (23) [100]	(0) 1,4 ± 0,9 (3) [100]
CF_2	(0) 41,6 ± 184,8 (1019) [100]	(0) 0,8 ± 0,9 (2) [100]
CF_3	(1) 8,2 ± 16,1 (81) [100]	(0) 1,2 ± 0,9 (3) [100]
CF_4	(0) 12,9 ± 36,4 (203) [100]	(0) 1 ± 0,9 (2) [100]
CF_5	(0) 10,5 ± 20 (107) [100]	(0) 1,3 ± 0,8 (2) [100]
CF_6	(2) 6,1 ± 8,4 (45) [100]	(0) 1 ± 0,8 (2) [100]
CF_7	(0) 18,6 ± 34,3 (154) [100]	(0) 1,4 ± 0,9 (3) [100]
CF_8	(0) 8,1 ± 11,5 (58) [100]	(0) 1,3 ± 1 (3) [100]
CF_9	(1) 8,3 ± 13,6 (59) [100]	(0) 0,9 ± 0,9 (2) [100]
CF_{10}	(0) 7,4 ± 15 (82) [100]	(0) 1,4 ± 0,9 (3) [100]
CF_{11}	(0) 4,8 ± 5 (18) [100]	(0) 1,2 ± 1 (3) [100]
CF_{12}	(2) 11,5 ± 17,5 (74) [100]	(0) 1,2 ± 1 (3) [100]
CF_{13}	(0) 7,6 ± 16,5 (92) [100]	(0) 1,2 ± 0,9 (2) [100]
CF_{14}	(2) 43,8 ± 208,6 (1148) [100]	(0) 1,4 ± 1 (3) [100]
CF_{15}	(0) 10,1 ± 17,6 (86) [100]	(0) 1,4 ± 0,9 (3) [93,3]
CF_{16}	(0) 9,5 ± 16,2 (68) [100]	(0) 1,2 ± 0,9 (3) [100]
CF_{17}	(2) 23,9 ± 58,5 (322) [100]	(0) 1,1 ± 1,1 (3) [100]
CF_{18}	(0) 50,5 ± 209,9 (1158) [100]	(0) 1,4 ± 0,8 (3) [100]
CF_{19}	(1) 14,2 ± 19,4 (81) [100]	(0) 1,3 ± 0,9 (3) [100]
CF_{20}	(2) 7,8 ± 7,1 (25) [100]	(0) 1,2 ± 0,9 (2) [100]
CF_{21}	(1) 13,4 ± 23,5 (75) [100]	(0) 1,4 ± 1 (3) [100]
CF_{22}	(2) 120,8 ± 595,4 (3272) [100]	(0) 1,4 ± 0,9 (3) [100]
CF_{23}	(2) 12,5 ± 16,4 (67) [100]	(0) 1,3 ± 0,8 (2) [100]
CF_{24}	(1) 11,2 ± 18,9 (85) [100]	(0) 1,1 ± 1 (3) [100]

Table 7b. Number of Backtracks solving Latin Square problem with different strategies

Strategy	LS n=5	LS n=6	LS n=7	LS n=9
S_1	0	0	9	18
S_2	9	163	>20679	>23860
S_3	0	0	0	0
S_4	0	0	9	18
S_5	0	0	9	18
S_6	9	163	>20679	>23860
S_7	0	0	0	0
S_8	0	0	9	18
	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]	(min) avg ± sd (max) [%]
Random	(0) 0 ± 0,2 (1) [100]	(0) 0,1 ± 0,3 (1) [100]	(0) 46,9 ± 251,3 (1377) [100]	(0) 181,1 ± 565,7 (2342) [93,3]
CF_1	(0) 1,4 ± 3,2 (9) [100]	(0) 8,9 ± 32,6 (163) [100]	(0) 1,3 ± 3,3 (9) [90]	(0) 3,6 ± 6,7 (18) [83,3]
CF_2	(0) 1,8 ± 3,6 (9) [100]	(0) 17,7 ± 44,9 (163) [100]	(0) 10,3 ± 45 (235) [90]	(0) 2,1 ± 5 (18) [83,3]
CF_3	(0) 0 ± 0 (0) [100]	(0) 7,8 ± 32 (163) [100]	(0) 2,2 ± 3,9 (9) [96,7]	(0) 6 ± 7,6 (18) [86,7]
CF_4	(0) 0,8 ± 2,5 (9) [100]	(0) 0 ± 0 (0) [100]	(0) 0,7 ± 2,4 (9) [86,7]	(0) 4,4 ± 7,2 (18) [93,3]
CF_5	(0) 0,9 ± 2,7 (9) [100]	(0) 21,2 ± 49,5 (163) [100]	(0) 3,7 ± 4,5 (9) [96,7]	(0) 1,4 ± 4,8 (18) [90]
CF_6	(0) 0,9 ± 2,7 (9) [100]	(0) 15,1 ± 43,5 (163) [100]	(0) 2,4 ± 4 (9) [90]	(0) 4,8 ± 7,2 (18) [90]
CF_7	(0) 1,2 ± 3,1 (9) [100]	(0) 24,1 ± 53,3 (163) [100]	(0) 2,5 ± 4,1 (9) [83,3]	(0) 5,7 ± 8,2 (18) [90]
CF_8	(0) 1,4 ± 3,3 (9) [100]	(0) 23,4 ± 52,2 (163) [100]	(0) 11,2 ± 45,4 (242) [93,3]	(0) 45,8 ± 176,9 (837) [73,3]
CF_9	(0) 0,9 ± 2,7 (9) [100]	(0) 2,1 ± 9,6 (52) [100]	(0) 285,7 ± 1355,8 (6505) [76,7]	(0) 7,3 ± 8,9 (19) [76,7]
CF_{10}	(0) 1,4 ± 3,2 (9) [100]	(0) 1,8 ± 9,7 (53) [100]	(0) 301 ± 1527,7 (7791) [86,7]	(0) 2,7 ± 5,8 (18) [90]
CF_{11}	(0) 0,6 ± 2,3 (9) [100]	(0) 18,4 ± 50 (163) [100]	(0) 1,8 ± 3,4 (9) [86,7]	(0) 4,1 ± 7 (18) [76,7]
CF_{12}	(0) 1,7 ± 3,4 (9) [100]	(0) 12,5 ± 41,8 (163) [100]	(0) 1,8 ± 3,6 (9) [73,3]	(0) 4,6 ± 7,6 (19) [80]
CF_{13}	(0) 0,8 ± 2,5 (9) [100]	(0) 3,6 ± 19,9 (109) [100]	(0) 1,8 ± 3,6 (9) [86,7]	(0) 3,6 ± 7,1 (18) [90]
CF_{14}	(0) 1,2 ± 3,1 (9) [100]	(0) 81,9 ± 409,7 (2245) [100]	(0) 74,7 ± 377,4 (1963) [90]	(0) 7,9 ± 11,4 (47) [86,7]
CF_{15}	(0) 10,2 ± 48 (264) [100]	(0) 4,7 ± 18,1 (82) [100]	(0) 1,1 ± 2,9 (9) [90]	(0) 5,3 ± 8,2 (18) [93,3]
CF_{16}	(0) 0,9 ± 2,6 (9) [100]	(0) 45 ± 209,3 (1141) [100]	(0) 2,1 ± 3,8 (9) [90]	(0) 7,3 ± 8,4 (18) [93,3]
CF_{17}	(0) 0,3 ± 1,6 (9) [100]	(0) 14,7 ± 43 (163) [100]	(0) 1,7 ± 3,5 (9) [86,7]	(0) 68,6 ± 300,2 (1445) [76,7]
CF_{18}	(0) 1,1 ± 2,8 (9) [100]	(0) 24,1 ± 56,9 (163) [100]	(0) 3,9 ± 6,9 (30) [80]	(0) 40,1 ± 125,3 (614) [80]
CF_{19}	(0) 0,9 ± 2,7 (9) [100]	(0) 6,8 ± 30,3 (163) [100]	(0) 5 ± 7 (26) [80]	(0) 7,1 ± 11,1 (40) [83,3]
CF_{20}	(0) 1,2 ± 3,1 (9) [100]	(0) 5,5 ± 23,1 (120) [100]	(0) 11,7 ± 34,3 (132) [93,3]	(0) 10,8 ± 27 (140) [90]
CF_{21}	(0) 1 ± 2,7 (9) [100]	(0) 16,3 ± 49,7 (163) [100]	(0) 8,6 ± 38,1 (187) [80]	(0) 4,4 ± 7,6 (18) [73,3]
CF_{22}	(0) 0,4 ± 1,5 (6) [100]	(0) 14,1 ± 42,3 (163) [100]	(0) 3 ± 6,2 (26) [86,7]	(0) 31,7 ± 114,3 (554) [76,7]
CF_{23}	(0) 1 ± 2,5 (9) [100]	(0) 8,7 ± 31,8 (163) [96,7]	(0) 0,7 ± 2,5 (9) [83,3]	(0) 7,7 ± 10,8 (40) [80]
CF_{24}	(0) 1,2 ± 3,1 (9) [100]	(0) 1,4 ± 7,5 (41) [100]	(0) 2,2 ± 3,8 (9) [83,3]	(0) 4,6 ± 7,5 (18) [83,3]

Table 7c. Number of Backtracks solving Knight Tour problem with different strategies

Strategy	KTP n=5
S_1	767
S_2	>42889
S_3	767
S_4	>22868
S_5	767
S_6	>42889
S_7	767
S_8	>22869
	(min) avg ± sd (max) [%]
Random	(10364) 15481,2 ± 6625,1 (26518) [16,7]
CF_1	(767) 10217,6 ± 17044,7 (48500) [23,3]
CF_2	(40) 3231,3 ± 3350,7 (8156) [23,3]
CF_3	(620) 9770,8 ± 11860,3 (27083) [13,3]
CF_4	(4) 6158,7 ± 11205,9 (29680) [23,3]
CF_5	(46) 4572,8 ± 3648,4 (8498) [26,7]
CF_6	(135) 4395,5 ± 3500,4 (7415) [13,3]
CF_7	(4) 5466,3 ± 5507 (15950) [33,3]
CF_8	(742) 6921,3 ± 3950,5 (12517) [20]
CF_9	(756) 7040,8 ± 4014,3 (15950) [33,3]
CF_{10}	(34) 6987,1 ± 8329,5 (23200) [36,7]
CF_{11}	(767) 6880,1 ± 4277,2 (15987) [43,3]
CF_{12}	(18) 7171,7 ± 8129,1 (26358) [30]
CF_{13}	(172) 4324,1 ± 3920,5 (10942) [33,3]
CF_{14}	(161) 7409,6 ± 10352,6 (41950) [46,7]
CF_{15}	(4) 4378 ± 4229,5 (12215) [30]
CF_{16}	(332) 7319 ± 8659,6 (32753) [40]
CF_{17}	(4) 7178,4 ± 10299,7 (30262) [26,7]
CF_{18}	(41) 2080,5 ± 2681,5 (7398) [20]
CF_{19}	(574) 4500,1 ± 5252,9 (15274) [30]
CF_{20}	(787) 9105,9 ± 8326,1 (24091) [30]
CF_{21}	(2155) 4836,3 ± 2715,6 (8498) [13,3]
CF_{22}	(27) 347 ± 452,5 (667) [6,7]
CF_{23}	(370) 4441,5 ± 4684,6 (8499) [13,3]
CF_{24}	(4) 11357,3 ± 14112,6 (29303) [13,3]

A global view of the search process can be obtained, by measuring its performance by means of some indicators, but such techniques do not take into account all the possible features. Many others features could be considered, but they interact in such unpredictable ways that it is often difficult to specify an adequate form of combining them. Our work was able to combine dynamically some basic enumeration strategies using the information of some few indicators of the search.

We used the number of backtracks as a common measure of search effort, it is a

good measure for the quality of both constraint propagation and enumeration. The CPU runtime measures are quite inaccurate, moreover the resulting time measure is consistent with the number of backtracks [3]. Considering that for competitions, the number of instances solved before timeout is the most important performance indicator, our proposal produce favourable results, as can be seeen in tables showing the number of backtracks for Magic square, Latin square and Knight tour where many of the basic strategies reach the timeout and the percentage of our experiments performed before the timeout is high (see [%] in Tables 6b, 7b, 7c). Then, our approach improves the percentage of resolution of the solver with a fixed strategy.

Backtracks are the standard counting in most constraint programming environments, so it facilitates the comparison of the search cost between different methods [21]. Clearly the measures were able to discriminate between the different strategies. The minimum value of backtracks reached in many experiments using our approach was zero (see **min** in Tables 6a, 6b, 7a, 7b and 7c. This demonstrates that it is possible to design "the best strategy" (or combination of enumeration strategies) for a specific problem.

4. Conclusions

In this paper we design a dynamic selection mechanism of enumeration strategies based in the information of the solving process. We are interested in dynamically detecting bad decisions concerning split strategies during resolution. We evaluate the efficiency of running strategies, and replace the ones showing bad results. We define some measures of the solving process, this information is analyzed to draw some indicators that are used to update the priority of application of enumeration strategies. The experimental results show that our proposal is able to consistently satisfy our requirements (of finding solutions well on average for a set of problems). We are at least in top-3 ranking finding good dynamic ways to solve many problems using a combination of enumeration strategies.

Among the main contributions of this work we can state the design and implementation of a solver that is able to measure the search process (using some basic indicators) in order to perform an on-the-fly replacement of enumeration strategies (using a portfolio of basic enumeration strategies). The solver is based on enumeration strategies of different natures (based on the size of variable domains, on the number of occurrences of the variables in the constraints) and some indicators on the resolution progress (backtracks, visited nodes, variables fixed, shallow backtracks, deep of the search tree, *etc.*). These basic components work properly together using our framework.

On the other hand, we have shown the development of a hybrid solver. In our approach the replacement of the enumeration strategies is performed depending on a quality rank (priority), which is computed by means of a choice function based hyperheuristic and its parameters are fine-tuned by a genetic algorithm. This tuning is a form of search too. We were focused on a general meta level tuning of the adaptation mechanism. This could be seen as the resolution of an optimization problem whose solution would contain the optimal configuration of the choice function.

From the point of view of the overall performance, considering that for competitions the number of instances solved before timeout is the most important indicator, our proposal outperforms many of the fixed enumeration strategies (Magic Square problem can not be solved by 5/8 strategies, Latin Square by 2/8 and Knight Tour by 4/8).

The correct tuning of the choice function weights has a crucial effect on the ability of the solver to properly solve specific problems. This setting is required because every problem has different characteristics. Parameter (choice function weights) tuning was difficult to achieve because the parameters are problem dependent and the best values of parameters are not stable along the search. Therefore static weights lead to suboptimal searches [15]. Moreover, parameters usually interact in a complex way, so a single parameter will have a different effect depending on the value of the others [15]. Then, combining the indicators in other form (non linear) could be an extension of this work. Additionally, this work may be extended implementing other choice functions and tuning their parameters with other metaheuristics, benefiting from the addition of restarting, considering constraint propagation and tackling optimization problems.

References

[1] Barták R., Rudová H., *Limited assignments: A new cutoff strategy for incomplete depth-first search.*, in H. Haddad, L. M. Liebrock, A. Omicini, R. L. Wainwright, editors, *SAC*, pp. 388–392, ACM, 2005.

[2] Beck J. C., Prosser P., Wallace R. J., *Toward understanding variable ordering heuristics for constraint satisfaction problems*, *Fourteenth Irish Artificial Intelligence and Cognitive Science Conference (AICS)*, pp. 11–16, 2003.

[3] Bessière C., Zanuttini B., Fernandez C., *Measuring search trees*, *Proceedings ECAI'04 Workshop on Modelling and Solving Problems with Constraints*, pp. 31–40, IOS Press, 2004.

[4] Borrett J. E., Tsang E. P. K., Walsh N. R., *Adaptive constraint satisfaction: The quickest first principle*, in W. Wahlster, editor, *ECAI*, pp. 160–164, John Wiley and Sons, Chichester, 1996.

[5] Boussemart F., Hemery F., Lecoutre C., Sais L., *Boosting systematic search by weighting constraints*, in R. L. de Mántaras, L. Saitta, editors, *ECAI*, pp. 146–150, IOS Press, 2004.

[6] Burke E. K., Kendall G., Hart E., Newall J., Ross P., Schulenburg S., *Handbook of Meta-heuristics*, chapter *Hyper-heuristics: An Emerging Direction inModern Search Technology*, pp. 457–474, Kluwer, 2003.

[7] Castro C., Monfroy E., Figueroa C., Meneses R., *An approach for dynamic split strategies in constraint solving*, in A. F. Gelbukh, A. de Albornoz, H. Terashima-Marín, editors, *MICAI*, vol. **3789** of *LNCS*, pp. 162–174, Springer, 2005.

[8] Chenouard R., Granvilliers L., Sebastian P., *Search heuristics for constraint-aided embodiment design*, *AI EDAM*, **23**(2):175–195, 2009.

[9] Crawford B., Castro C., Monfroy E., *Using a choice function for guiding enumeration in constraint solving*, *9th Mexican International Conference on Artificial Intelligence, MICAI 2010, Pachuca, Mexico, November 8–13, 2010, Special Sessions, Revised Papers*, pp. 37–42, 2010.

[10] Crawford B., Soto R., Montecinos M., Castro C., Monfroy E., *A framework*

for autonomous search in the eclipse solver, *24th IEA/AIE International Conference*, Lecture Notes in Computer Science, Syracuse, USA, 2011, Springer.

[11] Hamadi Y., Monfroy E., Saubion F., *What is autonomous search?*, Technical Report MSR-TR-2008-80, Microsoft Research, 2008.

[12] Bayardo Jr. R. J., Miranker D. P., *An optimal backtrack algorithm for tree-structured constraint satisfaction problems*, Artif. Intell., **71**(1):159–181, 1994.

[13] Liberatore P., *On the complexity of choosing the branching literal in dpll*, Artif. Intell., **116**(1–2):315–326, 2000.

[14] Mackworth A. K., Freuder E. C., *The complexity of some polynomial network consistency algorithms for constraint satisfaction problems*, Artif. Intell., **25**(1):65–74, 1985.

[15] Maturana J., Saubion F., *From parameter control to search control: Parameter control abstraction in evolutionary algorithms*, Constraint Programming Letters, **4**(1):39–65, 2008.

[16] Monfroy E., Castro C., Crawford B., *Adaptive enumeration strategies and metabacktracks for constraint solving*, in *ADVIS*, pp. 354–363, 2006.

[17] Petrovic S., Epstein S. L., Wallace R. J., *Learning a mixture of search heuristics*, *Proceedings of CP-07 Workshop on Autonomous Search, Providence, RI*, 2007.

[18] Sadeh N. M., Fox M. S., *Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem*, Artif. Intell., **86**(1):1–41, 1996.

[19] Sturdy P., *Learning good variable orderings*, in F. Rossi, editor, *CP*, vol. **2833** of *LNCS*, p. 997. Springer, 2003.

[20] Talbi E.-G., *Metaheuristics: From Design to Implementation*, Wiley Publishing, 2009.

[21] van Beek P., *Handbook of Constraint Programming*, chapter *Backtracking Search Algorithms*, Elsevier, 2006.

[22] Vidotto A., Brown K. N., Beck J. C., *Robust constraint solving using multiple heuristics*, in P. van Beek, editor, *CP*, vol. **3709** of *Lecture Notes in Computer Science*, p. 871. Springer, 2005.

[23] Wallace R. J., Grimes D., *Experimental studies of variable selection strategies based on constraint weights*, J. Algorithms, **63**(1–3):114–129, 2008.